A MATHEMATICAL APPROACH TO MODELLING THE FLOW OF DATA AND CONTROL IN COMPUTATIONAL NETWORKS

By

Lennart Johnsson and Danny Cohen

5120:TM:81

Appears in Carnegie-Mellon University Conference on
VLSI Systems and Computations
Edited by H.T. Kung, Bob Sproull, Guy Steele
October 19-21, 1981

A Mathematical Approach to Modelling the flow of Data and Control in Computational Networks

by

Lennart Johnsson
Caltech / ISI

Danny Cohen
ISI / Caltech

Abstract

This paper proposes a mathematical formalism for the synthesis and qualitative analysis of computational networks that treats data and control in the same manner. Expressions in this notation are given a direct interpretation in the implementation domain. Topology, broadcasting, pipelining, and similar properties of implementations can be determined directly from the expressions.[1]

This treatment of computational networks emphasizes the space/time tradeoff of implementations. A full instantiation in space of most computational problems is unrealistic, even in VLSI (Finnegan [4]). Therefore, computations also have to be at least partially instantiated in the time domain, requiring the use of explicit control mechanisms, which typically cause the data flow to be nonstationary and sometimes turbulent.

Introduction

The evaluation of mathematical expressions in general requires arithmetic operations as well as communication of data and control signals. The computations may be arranged in several ways, spanning the spectrum from fully parallel to fully sequential. The former approach spreads the computations in space whereas the latter spread them in time.

High computation throughput is typically achieved by spreading the computations entirely in space. The data flow is stationary and often laminar. Such a flow is easy to understand and to describe graphically.

Unfortunately, the cost of a full instantiation in space is often prohibitive for large problems. Therefore, the alternative instantiation in time has to be considered.

The spread in time requires a more complicated control mechanism that is needed for the fully parallel implementations. When computations are spread fully in space, the control of the operations may also be in the space domain (e.g., the size of an array may determine its computational response). On the other hand, with spread in time the control may often be in the time domain too. In addition, the data flow is typically not laminar, and turbulence (in the form of loops) occurs.

---

[1]Mailing addresses: Computer Science, MS-280, Caltech, Pasadena, California 91125 and ISI, 4676 Admiralty Way, Marina del Rey, California 90291.

Typically, control signalling varies between the parallel and the sequential regimes, just as with operations on data. Therefore, it is possible to use the same mathematical tools for treating the control and data flow through networks.

This work treats time/space tradeoffs mathematically. The required computations are spread in several ways in order to optimize the computational throughput of the implementations at a reasonable cost.

For many problems, arithmetic operations have to be performed simultaneously, such as those expressed by the mathematical symbols "sigma" for addition and "pi" for multiplication. The direct circuit implementation of simultaneous operations has lower throughput than sequential additions. Therefore, pipelined structures are generally sought.

The transformation of certain networks into pipelined structures affects the nature of the propagation of both data and control. These effects are studied and treated formally.

The organization of VLSI arrays is an active area of research. For example, systolic arrays for a few typical problems in numerical linear algebra and for the Discrete and Fast Fourier Transforms have been proposed and described by H.T. Kung [8], as has a two–dimensional array for solving a linear system of equations by S.Y. Kung [9]. The key issues in VLSI design are discussed in Mead and Conway [10].

To the best of our knowledge no proof of the correctness of these systems is available.

Two–dimensional arrays for the multiplication of band matrices have been described formally by Weiser and Davis [13].

All of the above systems use implicit control.

The computations of a Finite Impulse Response (FIR) filter and of the Discrete Fourier Transform (DFT) are used to illustrate the concept of mathematical modelling of both data flow and explicit control.

Finite Impulse Response Filters

The delay operator, Z, defined by $Zx(n) = x(n-1)$, is used for modelling storage (delay) elements. The properties of this operator and examples of using it for the synthesis and analysis of computational networks may be found in Cohen [1] and in Johnsson et al. [7].

There exists a straightforward correspondence between mathematical expressions and networks that contain only combinational logic and delay (memory) units.

With symbolic manipulation and operator calculus it is possible to generate for a given expression (and the corresponding network) several alternatives that compute the same results while having different properties like throughput and delay.

The transformations between computationally equivalent networks and their evaluation with respect to a given set of criteria may be carried out formally, in a way which may eventually be automated.

For example, a general FIR filter (Oppenheim and Schafer [11], Rabiner and Gold [12], and Cohen [1]) is defined by:

$$y(n) = \sum_{i=0}^{N-1} a(i)x(n-i) \qquad (1)$$

which may be expressed as

$$y(n) = \sum_{i=0}^{N-1} a(i)\, Z^{i}\, x(n) \qquad (2)$$

or as

$$y(n) = \sum_{i=0}^{N-1} Z^{i} \left(a(i)x(n) \right) \qquad (3)$$

Eventhough (2) and (3) are computationally equivalent, they have different characteristics. For example, it is clear that (2) requires a simultaneous addition of N elements, whereas (3) uses pipelined additions which may be performed within a shorter cycle than needed for the former, hence yielding a higher throughput.

It is also clear that (3) requires the broadcasting of the input signal $X = \{x(n)\}$ to all the computation elements, unlike (2), which does not require it.

Figures 1 and 2 show the networks corresponding to these expressions.

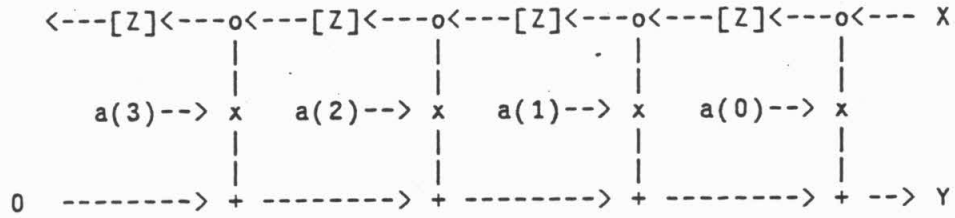

Figure 1: The implementation of (2)

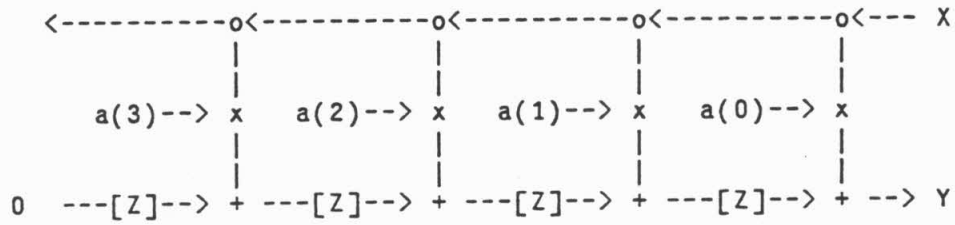

Figure 2: The implementation of (3)

Both of these networks spread the computations entirely in space, thereby achieving fully parallel implementations. In the network of Figure 1 all the computations for each y(n) are simultaneous whereas in that of Figure 2 the computations are skewed in the time/space domain, which is typical of pipelines.

Both of these implementations have implicit control embedded in space, by virtue of the array size, N.

By explicitly addressing the control issue (Cohen and Tyree [2]) these networks can be

transformed into sequential implementations with significantly different characteristics, not only of performance but also regarding issues of error containment and hardware reduction.

As a result, the mathematical requirement to operate on all the input elements, {x(n)}, with all the constant coefficients, {a(i)}, (implemented in the parallel network by moving the data along the stationary coefficients), is implemented in the sequential network by the dual approach of (relatively) stationary input elements and moving coefficients.

An explicit control signal, U, is used to implement the accumulation of a variable number, N, of elements, where N is determined only at run time. The expression U'A + UB, which is equal to A when U = 0 and to B when U = 1, is used to model a multiplexer[2] for the variable length accumulation, as shown in Figure 3. Note that since the value 0 is connected to the 1–input of the multiplexer the effect of U = 1 is to clear (reset) the accumulator (the Z–element).

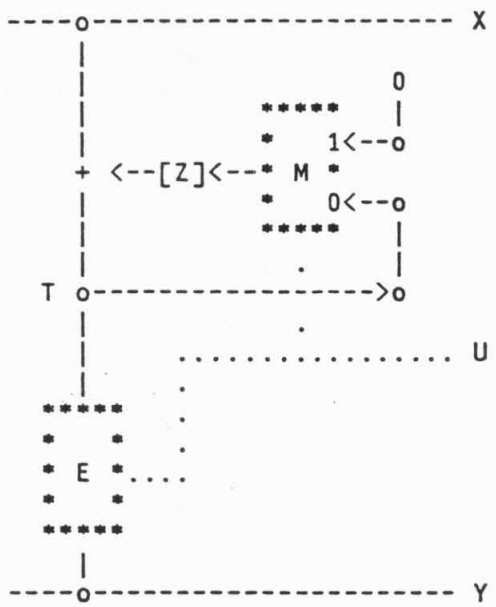

Figure 3: A finite accumulation network

The function performed by the network in Figure 3 is described by the following expression:

$$Y = UT \quad \text{where} \quad T = X + Z(U'T + U0) = X + ZU'T \qquad (4)$$

The UT means "enabling" Y to have the value T when U = 1 and not having any value when U = 0 (e.g., E in Figure 3 is a tristate driver). Hence, the explicit control that resets accumulators (the Z–units) to 0 also enables the E–units to drive the output bus. Note that the existence of the output of E is controlled by U, not its value, as in the case of multiplexers.

The output of this network, Y, is always the accumulative sum of all the input values, X, since the last time U = 1.

The right–hand side of equation (4) can be expanded a few times to make the nature of the network more explicit.

---

[2] A is called the 0–input and B is the 1–input.

$$
\begin{aligned}
T &= X + ZU'T = \\
&= X + ZU'(X + ZU'T) = \\
&= X + ZU'\ X + ZU'ZU'(X + ZU'T) = \\
&= X + ZU'\ X + ZU'ZU'\ X + ZU'ZU'ZU'(X + ZU'T) = \ldots .
\end{aligned}
\tag{5}
$$

The expansion can be carried on indefinitely. However, if U = 1 every N cycles, then at most N–1 consecutive values of U' may be equal to 1. When U = 1 the output Y is equal to the sum of the last N input values.

To simplify the expressions for the case with cyclic control, the following notation is introduced. Let [k] be the remainder of k when divided by N, and let $\langle k \rangle = k-[k]$, which is the largest multiple of N that does not exceed k. Hence, $k = \langle k \rangle + [k]$ for all values of k.

A signal, U, having the value 1 every N cycles and the value 0 in between is expressed by U(t) = d([t–j]), where d(0) = 1 and d(x≠ 0) = 0. This j is the "phase" of the control signal.

If the phase of U is j, then the phase of ZU is j + 1, because ZU is delayed one cycle behind the U signal[3].

The expression above, (5), for the time k, is rewritten as

$$
Y(t=k) = \left(\sum_{m=0}^{[k-j]} Z^{m} \right) X \tag{6}
$$

Only when [k–j] = N–1 which is when [k] = [j–1] the output is the sum of the last N input values.

Thie network shown in Figure 3 produces such an N–term sum only once in every N cycles. If such a sum is needed for every cycle, then N units are required. These units should be controlled by U's of different phases.

In equation (3) the {a(i)} were assumed to be time independent, therefore not affected by the Z–operator. However, one may arrange a network where the {a(i)} travel in a circular shift register of length N, such that the a(i) coefficient follows, rather than precedes, the a(i + 1) coefficient. In this case Za(i) = a(i + 1) and $a(i) = Z^{i}a(0)$. This arrangement of coefficients is called "dynamic coefficients."

By introducing the notions of controlled selection (multiplexing), finite accumulation, and dynamic coefficients, it is possible to obtain implementations consisting of an array of sequential elements, as shown in Figure 4. Such arrays effectively perform a cyclic convolution.

The network in Figure 4 computes

$$
Y = \sum_{i=0}^{M} \left[(Z^{i}U) \left(\sum_{m=0}^{B} Z^{i} A Z^{m} X \right) \right] \quad \text{where} \quad B = [k-i-1] \tag{7}
$$

---

[3] This is proven by ZU(t) = d([t–1–j]) = d([t–(j + 1)]).

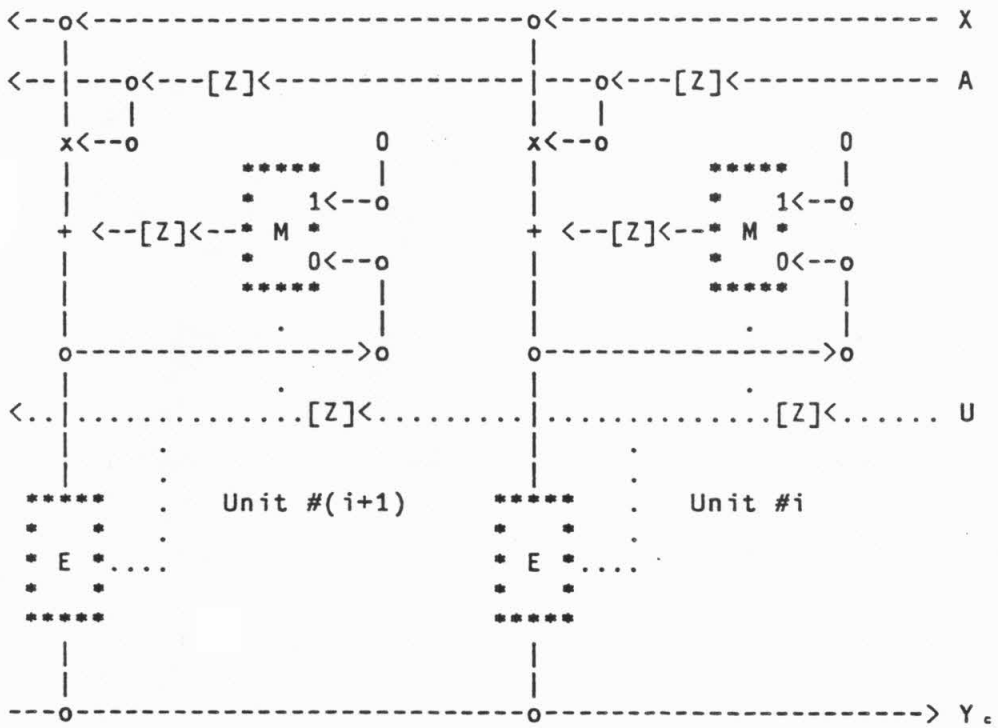

Figure 4: Two sequential elements for an FIR network

Note that the outer "sigma" and the Z^iU represent the selection of that output of the ith unit exactly when the output of that unit is the sum of the last N products of x's and a's. Hence it expresses the same function[4] as equations (1), (2), and (3), and the network in Figure 4 is functionally equivalent to the networks in Figures 1 and 2.

The implementation represented in Figure 4 requires extra circuitry associated with the explicit control. The previous implementations do not need the extra circuitry because the value of N is built implicitly into their configurations. Another nice aspect of those networks is that the data flow inside them is always laminar and easy to track, unlike the turbulent flow in the network of Figure 4.

However, the sequential implementation has some important advantages deserving attention.

In networks that are fully instantiated in space, every module (multiplier, adder, and delay) and every connection are directly involved in the computation of every y(k). Hence, any component failure in the entire network causes every output to be erroneous. On the other hand, in the network of Figure 4, failure of a single adder or a single accumulator (a Z–element) affects only 1/N of the output set.

If the number of the required {y(k)} is smaller than the number of {x(k)}, as for down–sampling by a factor of K, the hardware can be reduced by using only N/K units, instead of N as needed for the networks of Figures 2 and 3. Down– sampling by those

---

[4] Proof: Unit #i has the phase of i and is selected (enabled) when [k] = i, at that time B = [k–i–1] = [i–i–1] = N–1. Hence, Y is a sum of the last N terms.

networks can be done only by discarding K–1 out of every K output values, not by reducing the computation, as in the network of Figure 4.

From the signal processing point of view this computation scheme is better than first down–sampling and then filtering.

There is no way to use any of the former networks for lengths greater than the original N. The latter network, with its explicit control, may easily be modified to handle shorter lengths than the original N, which is the number of units in the configuration. If a longer length, M, has to be handled, the latter network can still be used. It will produce only some (N/M) of the {y(k)}. This is better than the former networks, which cannot produce any of the {y(k)}.

Even with the extra hardware, the latter network has several advantages over the former ones.

The Discrete Fourier Transform

A more interesting case is the implementation of the continuous DFT computation (Rabiner and Gold [12], Johnsson and Cohen [6]). A continuous DFT is a DFT that is updated every input cycle. If computed on windows of length N, then the straightforward implementation (as a matrix multiplication) requires NM complex mutiplications, where M is the number of the required components of the DFT. The use of the FFT (Cooley and Tukey [3]) requires only NlogN/2 steps, independent of the value of M.

Fully parallel implementations of any interesting DFT computation require more components than it is practical to consider.

A formal treatment of FFT arrays has been presented in Johnsson and Cohen [5]. Several networks for the DFT can be derived formally from its definition. One sample implementation is derived and shown here.

The implementation resulting from our formal treatment of the DFT requires only 2M (or possibly M) multipliers. (Compared this with NM multipliers for a full instantiation in space and NlogN/2 multipliers for the FFT.)

The mth DFT component at time t = n is defined by

$$y(m,n) = \sum_{j=0}^{N-1} w^{jm} x(n-j) \qquad (8)$$

which is rewritten as

$$y(m,n) = \sum_{j=0}^{N-1} w^{jm} Z^{j} x(n) = \sum_{j=0}^{N-1} (w^{m} Z)^{j} x(n) \qquad (9)$$

Since this is a geometric series its sum is

$$y(m,n) = \sum_{j=0}^{N-1} (w^{m} Z)^{j} x(n) = (1 - w^{mN} Z^{N})^{-1} (1 - wZ)^{-1} x(n) \qquad (10)$$

These two operators commute and may be combined ("multiplied") in either order. However, for response reasons the order used in (10) is preferred.

Eventhough theoretically $w^{mN} = 1$, it may be beneficial not to eliminate the multiplication by 1, in order to treat the effects of roundoff errors in limited precision implementations.

The implementation of $(1 - w^m Z)^{-1}$ may not look intuitive to the untrained reader. However, note that

$$U = (1 - w^m Z)^{-1} V \quad \text{which yields:} \tag{11}$$

$$(1 - w^m Z)\, U = V \quad \text{or} \quad U = w^m ZU + V$$

This equation is implemented by the network in Figure 5.

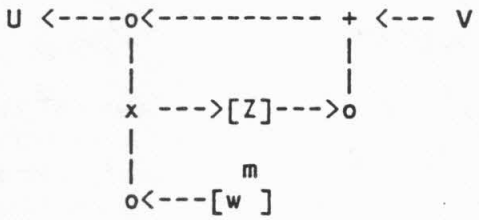

Figure 5: The implementation of $U = (1-w^m Z)^{-1} V$

With the network shown in Figure 5 for $(1-w^m Z)^{-1}$ the implementation of equation (10) is:

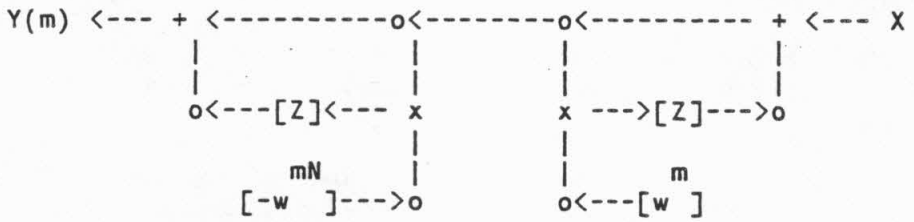

Figure 6: The continuous implementation of y(m,n)

Conclusions

A rigorous mathematical formal approach may be applied to the synthesis and the analysis of both the data paths and the control signalling of computational networks.

This mathematical tool is instrumental for the investigation of time/space tradeoffs, and supports the transformations of networks between implementations ranging from fully parallel to sequential or both, such as in pipelines.

The same formalism used for the manipulation of the data flow is also used for control signalling.

In addition to its value for the synthesis of networks, this formalism is also very useful for verification of correctness and qualitative analysis. Network properties such as pipelining and broadcasting can be determined directly from expressions in the notation.

Acknowledgments

The authors gratefully acknowledge the support for this research provided generously by the Defense Advanced Research Projects Agency under contract MDA-80-C-0523 with the USC/Information Sciences Institute and contract N00014-79-C-0597 with the California Institute of Technology.

Views and conclusions contained in this paper are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, nor any person or agency connected with them.

References

[1] Cohen, D., "Mathematical approach to iterative computational networks," Proceedings of the Fourth Symposium on Computer Arithmetic, pp. 226–238, October 1978, also published as USC/Information Sciences Institute RR–78–73, November 1978.

[2] Cohen, D., and V.C. Tyree, "VLSI system for Synthetic Aperture Radar (SAR) processing," Proceedings of the Society of Photo–Optical Instrumentation Engineers (SPIE), vol 186, pp. 166–177, 1979.

[3] Cooley, J.W., and J.W. Tukey, "An algorithm for machine calculation of complex fourier series," Mathematics of Computation, vol 19, pp. 297–301, 1965

[4] Finnegan, J. "The VLSI approach to computation complexity," in these proceedings.

[5] Johnsson, S.L., and D. Cohen, "Computational arrays for the Discrete Fourier Transform," COMPCON 81, February 1981.

[6] Johnsson, S.L., and D. Cohen, "A VLSI approach to real–time computation problems," Proceedings of the Society of Photo–Optical Instrumentation Engineers (SPIE), vol 298, September 1981.

[7] Johnsson, S.L., U. Weiser, D. Cohen and A. Davis, "Towards a formal treatment of VLSI arrays," Proceedings of the Second Caltech Conference on VLSI, January 1981.

[8] Kung H.T., and C.E. Leiserson, "Algorithms for VLSI processor arrays," in [10].

[9] Kung S.Y., "VLSI matrix computation array processor," The MIT Conference on Advanced Research in Integrated Circuits, February 1980.

[10] Mead C.A., and L. A. Conway, Introduction to VLSI Systems, Addison–Wesley, 1980.

[11] Oppenheim, A.V., and R.W. Schafer, Digital Signal Processing, Prentice–Hall, 1976.

[12] Rabiner, L.R., and B. Gold, Theory and Application of Digital Signal Processing, Prentice–Hall 1975.

[13] Weiser U., and A. Davis, "Mathematical representation for VLSI arrays," University of Utah, Computer Science Department, Report UUCS–80–111, September 1980.

List of Figures

Figure 1: The implementation of (2) 3
Figure 2: The implementation of (3) 3
Figure 3: A finite accumulation network 4
Figure 4: Two sequential elements for an FIR network 6
Figure 5: The implementation of $U = (1-w^mZ)^{-1}V$ 8
Figure 6: The continuous implementation of y(m,n) 8